Digital Signature Standard Validation System (DSSVS) User's Guide

(For Version 2.3 of the DSSVS Software Tool)

Original: June 20, 1997
Last Modified: May 13, 1999

Jim Foti
Information Technology Laboratory
National Institute of Standards and Technology

Table of Contents

1. INTRODUCTION

This User's Guide is intended to assist persons who are conducting validation tests for conformance to Federal Information Processing Standard Publication (FIPS PUB) 186, *Digital Signature Standard (DSS)*, and FIPS PUB 180-1, *Secure Hash Standard (SHS)*.

This section provides the purpose, design philosophy, and high-level description of the operation of the DSSVS.

1.1 Purpose of the DSSVS

The National Institute of Standards and Technology's Digital Signature Standard Validation System (DSSVS) tests message authentication and integrity devices for conformance to two data authentication and integrity standards: Federal Information Processing Standard Publication (FIPS PUB) 186, *Digital Signature Standard (DSS)*, and FIPS PUB 180-1, *Secure Hash Standard (SHS)*. The DSSVS is designed to allow a tester to perform testing on message authentication devices which are located remotely from the tester.

Note that an implementation might not incorporate both the SHS and DSS. Therefore, implementations will either be tested for the SHS only, or for *both* the DSS and SHS.

1.2 Design Philosophy

The DSSVS is designed to allow testing of message authentication devices from locations remote to the DSSVS. The DSSVS tests can be performed on a message authentication device at the remote site and the results communicated to the DSSVS in the form of response files.

NIST validation programs are conformance tests rather than measures of product security. NIST validation tests are designed to assist in detection of accidental implementation errors, and are not designed to detect intentional attempts to misrepresent conformance. Thus, validation by NIST should not be interpreted as an evaluation or endorsement of overall product security.

A message authentication device is considered validated for a test option when it passes the appropriate DSSVS tests. For each test that is passed, a notation will be made on the validation certificate. DSSVS testing is via statistical sampling, so validation of an option does not guarantee 100% conformance with the option in the standards.

Laboratories which are accredited by the National Voluntary Laboratory Accreditation Program (NVLAP) manage the testing process for the DSS and SHS. A laboratory is responsible for generating values to be used in testing the implementation, and then checking that the implementation's results are correct. It is up to the laboratory to determine whether the actual testing will be done at the laboratory or at the vendor's site. In the latter case, the vendor will be responsible for using the identified implementation to generate responses for all applicable tests.

The intent of the validation process is to provide rigorous conformance testing that can be performed at modest cost. NIST does not attempt to prevent a dishonest vendor from purchasing a validated implementation and validating that implementation as the vendor's own product. However, customers who wish to protect themselves against a dishonest vendor could require that the vendor revalidate the implementation in the customer's presence.

1.3 High-Level Operation of the DSSVS

In batch mode, the operation of the DSSVS is straightforward. A vendor submits an application to a laboratory to validate an implementation of the DSS and/or SHS. Included in the required information is an indication of the desired tests. Based on the information in the vendor's application, the DSSVS creates a series of ASCII files that contain the appropriate information for each test (with a separate set of files for each test). The information is supplied by the DSSVS to the DUT for each test in a "request" file, with ".req" as the extension. This file is then forwarded to the entity performing the test. The vendor translates the requests into a form that can be accepted by the device under test (DUT), enters the requests, collects and formats the corresponding results (termed "responses"), stores them in a series of "response" files in ASCII format, and forwards the response file to the DSSVS for checking. Each response file shall have the same filename as its corresponding request file, but with ".rsp" as the extension.

In the event of a failure to validate, retesting may be performed. In this case, some data from the previous set of tests may still be applicable. Thus, a vendor may wish to retain all data from a set of tests until receiving notice of the disposition of the tests.

1.4 Validation Test Completion

When validation testing is complete, the tester should generate a results file, using the menu item under "Utilities", which summarizes the results of each test performed for a product (e.g., "Signature Generation: 80 out of 80 signatures generated correctly", etc.). The generated results file has an ".out" extentsion. At this time the tester should provide NIST with the following information:

1. Name and Version Number of the Tested Implementation (DUT),
2. Vendor Name, and Point of Contact Information,
3. Processor and Operating System with which the DUT was tested, if the DUT is implemented in software,
4. Brief description of the DUT, or the product/product family in which the DUT is implemented by the vendor (2-3 sentences), and
5. Electronic copy of:

○ the Validation Information file,
○ all Request files,
○ all Facts files,
○ all Response files, and
○ the Results file, which summarizes the test results; the tester should be sure that this information accurately reflects the test results, since it will be used by NIST in determining whether or not a validation certificate will be issued.

The above information will be saved by NIST, and used to generate a validation certificate, which will be returned to the vendor via the tester. In addition, an entry will be added to the DSS/SHS Validation List (or the SHS Validation List, if only SHA-1 tests are performed) which is maintained by NIST.

2. DSSVS TEST DESCRIPTIONS

2.1 SHS Tests

There are three areas of the Secure Hash Standard for which the DSSVS provides conformance testing: messages of varying length, selected long messages, and pseudorandomly generated messages. Since it is possible for a DUT to correctly handle the hashing of byte-oriented messages (and not messages of a non-byte length), the SHS tests each come in two flavors.

2.1.1 Messages of Varying Length

An implementation of the SHS must be able to correctly generate message digests for messages of arbitrary length. The DSSVS tests this function by supplying the DUT with 1025 pseudorandomly generated messages with lengths from 0 to 1024 bits (for a DUT that only hashes byte-oriented data correctly, 128 messages of length 8, 16, 24,...,1024 bits will be supplied). The DSSVS concurrently generates its own message digests on those same messages supplied to the DUT, and stores those values. The DUT generates message digests for each message. The DSSVS then compares its stored message digests to the ones generated by the DUT. If all messages compare correctly, then this test is passed.

2.1.2 Selected Long Messages

The DSSVS tests for correct generation of selected long messages. A list of 100 messages, each of length > 1024, is supplied to the DUT, which then generates message digests for each. If the DUT only handles byte-oriented data correctly, then each of these messages is a multiple of 8 bits in length. The DSSVS regenerates the message digests and compares them to the ones generated by the DUT.

2.1.3 Pseudorandomly Generated Messages

The DSSVS tests the correctness of message digests generated from pseudorandomly generated messages by supplying a seed M of length 420 bits. The DUT uses this seed to generate 100 message digests using a specific procedure; the details of this procedure are in Appendix E. The DSSVS regenerates the message digests using the same procedure and compares each of the 100 message digests with those generated by the DUT.

2.2 DSS Tests

There are six areas of the DSS for which the DSSVS provides conformance testing: primality testing, generation of the parameters p, q, and g, verification of another implementation's correct generation of p, q, and g, generation of public/private key pairs (x,y), signature generation, and signature verification. For those tests that involve the generation of either a private key, x, or the pseudorandom signature value, k, the DSSVS allows for implementations which use various pseudorandom number generators. There are three such generators: two are found in Appendix 3 of FIPS 186, which use SHA-1 and DES, and a third is found in Appendix C of ANSI X9.17.

As mentioned above, the DSSVS provides a test for the verification of p, q, and g when these parameters have been generated elsewhere and are imported into a cryptographic module. Although the DSS does not specifically address importation of these parameters, it does require that their generation be accomplished using either Appendix 2 of the DSS or other FIPS-approved security methods. In order to guarantee that imported parameters have been generated correctly, this test is recommended by NIST.

For the modulus size $L = 512 + 64 * i$, $0 \leq i \leq 8$, there are 9 permissible discrete values, ranging from 512 ($i = 0$) to 1024 ($i = 8$). Although maximum interoperability between implementations of the DSS is assured if all 9 values are used, some implementations may only include a subset. Therefore, for the purpose of generality, it is left to the DUT vendor to indicate which subset of values are implemented. Each of the DSS tests allows the tester to generate parameters and request files for any combination of those modulus sizes.

The following subsections provide an overview of the DSS test areas.

2.2.1 Primality Test

The DSS does not require the use of a specific primality test. However, a robust test is required by the DSS for use in generating p and q using the routine in Appendix 2.2 of the DSS. The test given in Appendix 2.1 of the DSS is suitable for this purpose. Regardless of which primality test is used, an implementation should include a function prime(w) that accepts an integer w with $2 < w < 2^{LMAX}$ (where LMAX equals the maximum modulus size that can be handled by the DUT) and returns true if w is prime and false otherwise.

The DSSVS tests this function by supplying between 3 and 15 numbers per modulus size (depending on the mod sizes selected for testing - see table under “Primality Test” in Section 3.5), of which several values are selected to be non-prime values (as determined by using the primality test given in Appendix 2.1 of the DSS). It is left to the DUT determine the primality of each one of those numbers. The DUT returns its determination of the primality of each number to the DSSVS, which compares its own stored results with the DUT response.

2.2.2 Generation of p, q and g

The DSS requires that primes p and q be generated using a FIPS-approved method. Such a method is described in Appendix 2.2 of the DSS. The DSS also requires that any implementation be able to correctly generate the parameter g for a pair (p,q). Correctness of g can be determined by inspecting the parameter h used to generate g and verifying that $1 < h < p-1$ and $g = h^{(p-1)/q} \bmod p > 1$.

Based on the mod sizes selected for testing, the DSSVS determines how many pqg triples shall be tested for each mod size (see table under "Primality Test" in Section 3.5). The DUT then generates the required number of sextets (SEED, q, p, counter, h, g) for each modulus size to be tested, and sends the results to the DSSVS. The DSSVS checks the results by using the information in each of the sextets to recompute the pqg triple, using the method is described in Appendix 2.2 of the DSS. ***Note: This test cannot be performed if the DUT cannot provide the DSSVS with values for SEED, counter, and h.***

2.2.3 Key Generation for Private and Public Key Pairs

The private key x must be generated by a FIPS-approved method. At present, the only two such methods of generating x are the technique of Appendix 3.1 of the DSS and the use of the pseudorandom number generator specified in Appendix C of ANSI X9.17, "Financial Institution Key Management (Wholesale)". The DSS-based method can use either the SHA-1 (as described in Appendix 3.3 of the DSS) or the Data Encryption Standard (DES) (as described in Appendix 3.4 of the DSS) as a pseudorandom source to generate a 160-bit output. This pseudorandom source is required for the generation of x. Therefore, it is assumed that all implementations of the DSS will use either 1) DSS with SHA-1, 2) DSS with DES, or 3) the ANSI X9.17 method as the pseudorandom source. Depending on which pseudorandom source is selected, the information generated by the DUT will differ.

It is also assumed that the public key y, which is directly related to x, will be computed by the DUT.

For each modulus size selected, the DSSVS tests the generation of 10 values of x, by sending the DUT the modulus sizes and pseudorandom number generation method expected. For those implementations which can import pqg triples and use them for key generation, then the DSSVS will also provide the DUT with a pqg triple for each modulus size selected.

The DUT then generates the required x and y values, and passes them to the DSSVS. If the DUT is able to output the values XKEY and XSEED used in generating x, then these are also included in the response file, and the DSSVS checks each value of x through re-calculation. If XKEY and XSEED are not available, then only y can be re-calculated using x, g, and p.

2.2.4 Signature Generation

An implementation of the DSS must be able to correctly generate the (r,s) pairs that represent a digital signature. For each modulus size, the DSSVS supplies 10 messages to the DUT, along with a pqg triple and x value, assuming the DUT can import those values. The DUT uses previously generated or DSSVS-supplied parameters to generate the corresponding signatures and returns them to the DSSVS. If the internal parameter k used during signature generation can be exported, then this value, in addition to the signature, is returned to the DSSVS. If k is present, the DSSVS re-calculates the signature; if k is absent, then the DSSVS implicitly determines that the signature is correct by performing signature verification using the stored public key.

2.2.5 Signature Verification

The DSSVS tests implementations for the ability to recognize valid and invalid signatures. For each modulus size, the DSSVS generates 3 x and y key pairs. The number of pseudorandom messages signed using each x is based on the number of mod sizes selected for testing (see table under "Signature Verification Test" in Section 3.5). Some of these signatures are altered so that signature verification should fail. The messages, signatures, pqg triples, and y values are then forwarded to the DUT. The DUT then verifies the signatures and returns the results to the DSSVS, which compares these received results with its own stored results.

2.2.6 Verification of Correct Generation of p, q, and g

The DSS requires that the prime parameters p and q be generated by a FIPS-approved method. The only such method is that specified in Appendix 2.2 of the DSS. Therefore, if a cryptographic module accepts values of p, q, and g from another module, it is assumed that Appendix 2.2 of the DSS was used to generate those values. For each modulus size, the DSSVS supplies a specified number of sextets (SEED, q, p, counter, g, h) to the DUT - this number is based on the mod sizes selected for testing (see table under "Primality Test" in Section 3.5). Some of the values in some of the sextets are modified before being passed to the DUT. The DUT verifies the correctness of each sextet, and returns the results to the DSSVS, which compares these received results with its own stored results. Note that the implementation of a test for correct generation of these parameters is not required by the DSS. However, if a cryptographic module implements such a test, the DSSVS test will verify its accuracy.

3. DSSVS OPERATION

3.1 Concept

The DSSVS keeps track of each validation by storing relevant information, such as the product name, vendor, vendor address, chosen tests, and parameter settings in a validation information file with a ".inf" extension. Once all the information has been entered, the user can then select a test (or tests) to run, and generate request files that will ultimately be sent to the vendor; these files have names that correspond with the applicable test, but all of these files have a ".req" extension. At the same time that a request file is generated for the vendor, a "facts" file, with a ".fax" extension, is generated for use by the DSSVS during response file verification. It has the same name as the corresponding request file. When the DUT is used to generate test results, based on the information in the request file, a response file with a ".rsp" extension is constructed. This file is then used by the DSSVS, in conjunction with the corresponding facts file, to verify the correctness of the DUT's test results. Note that a DSSVS user can create a "test" response file in many cases by using information from the corresponding facts file, rather than using a separate implementation to generate values.

The validation information file (".inf") also contains status information for each of the different tests. This information tells the user if the test settings have been configured, if a request file has been generated, and whether the test results have been verified. A table with checked boxes conveys this information in the validation information window.

The validation information file also contains more detailed information about the test results, including the number of test values, number of tests passed/failed, etc.

3.2 Windows in the DSSVS Tool

There are several basic windows that are accessible in the DSSVS, and they are described in the following sections.

3.2.1 Main DSSVS Window

The Main DSSVS Window has two menus, "File" and "Utilities". The "File" menu has the following menu items:

- **New**: creates a new validation information (.inf) file,

- **Open**: opens an existing validation information (.inf) file,

- **Save**: saves the state of the current validation to disk in a validation information (.inf) file,

- **Save As**: saves the current validation in a file whose path and filename are specified by the user,

• **Validation Info**: displays the setup of the current validation, including vendor and implementation information and the status of the various tests.

• **Exit**: closes the DSSVS program. If a validation has been updated, but those changes have not been saved, then the program will prompt the user to save those changes, before exiting. *Note that if the application is closed using the "X" button on the title bar of the Main window, then the application will be exited* ***without saving any changes!***

The "Utilities" menu has the following menu items:

• **Generate Results File**: Based on results of verifying the various test results, this will generate a "results" file of the same name as the opened validation information file, but with an ".out" extension. A sample results file may be found in Appendix B.

• **Generate PQG**: Allows a user to generate a file, "pqg_file", that contains sets of p, q, g, h, counter, and seed. The user can have the tool generate from 1 to 10 pqg sets for any number of modulus sizes (512,..., 1024). As input, the user is asked to enter 40 hex characters as the initial seed. This task becomes more time-consuming as the modulus size increases. A message box displays how many pqg sets have been generated and requested, as well as the counter value for the currect pqg set generation. The user can also cancel this operation. If this is selected, then the current pqg set will complete its generation before the cancellation takes effect, and all of the pqg sets that were generated will be found in "pqg_file".

Note: A tester may cut and paste pqg sets from various files to create different pqg_files. But the tester is not required to use a different pqg_file for each validation. It is suggested, however, that new values be generated periodically, and that the pqg_file be modified accordingly.

• **Generate Key Pairs**: Generates a file, "keypairs.xy", that contains any number of sets of p, q, g, xkey, xseed, x, and y for all selected modulus sizes, based on one of three possible pseudorandom number generation methods (SHA-1, DES, ANSI X9.17). Values for p, q, and g are obtained by default from "pqg_file". Note that this is NOT the Key Generation Test. However, values generated with this option can be used to create an example file to test the Key Generation test of the tool.

• **Generate Seed Value**: Takes input characters, time between keystrokes, and the SHA-1 to generate a *seed* value. This can be invoked when (x,y) key pairs are being generated, when it is necessary to enter values for xkey and xseed. Note that this generates a 160-bit number in accordance with the FIPS-approved pseudorandom number generation method specified in FIPS 186, Appendix 3.1, using "G from SHA" in Appendix 3.3. When "Generate" is selected for certain DSS tests, and the tester is prompted to enter a value for XKEY or XSEED, a button can be selected which will invoke this "Generate Seed Value" window in order to generate a pseudorandom value for XKEY or XSEED.

3.2.2 Validation Information Window

When a validation information (.inf) file is selected - either with the “New” or “Open” menu item, the Validation Information Window appears. This can also be viewed by selecting the “Validation Info” menu item. This is where the tester enters management information for the validation. On the right side of the window, there is a “Status” matrix, which includes checkboxes for each test that indicate whether a test had completed the configuration, generation, or verification stage.

In the box for “filename”, the tester should indicate the path and filename for the validation information file. This filename should have an “.inf” extension. Once the “OK” button is selected, this window is closed, and the name of the opened validation information file is listed in the title bar of the Main DSSVS Window. At this time, the DSS Validation Suite Window is displayed within the Main DSSVS Window.

The “.inf” file is an ASCII file which can be manually edited, but it is best viewed and edited using an application such as Windows WordPad, as opposed to Windows NotePad or Edit or Emacs. A sample “.inf” file can be found in Appendix A.

Note: By default, a new file will be saved in the same directory in which the DSSVS executable is located. If a path name is specified for a file in another directory, you must first create that directory outside of the DSSVS application (the tool will not automatically do this for you). If you wish to use request, facts, and response files that are in that same directory as the information file, then click “OK” to close the Validation Information Window. Next, select “Open”, then find the validation information file that was just specified - this will set the selected directory, so that the tool will place request, facts, and output files in the same directory as the validation information file.

3.2.3 DSS Validation Suite Window

This window appears within the Main DSSVS Window when a validation information file is selected. It contains buttons for all seven possible tests (All three types of SHA-1 tests are handled by selecting the “SHA-1" button.). Selecting a button will display the Test Window for a particular test. These seven tests are listed in each of sections 3.4-3.6.

3.2.4 Test Window

This is used to individually handle each test. This window displays three buttons, “Configure”, “Generate”, and “Verify”, and gives a brief description of the test. It also displays three check boxes, showing which of the above functions have been completed. “Configure” allows the tester to select a) the modulus sizes for which the test will be run, b) bit- versus byte-oriented hashing, c) random number generation method, etc. Once the “OK” button has been selected in the configuration window, the configuration information is recorded in the “.inf” file and the corresponding check box is noted in the Test Window. **If any configuration information had been saved previously, then**

it will be overwritten! If the tester wishes to just view the old configuration information, and not overwrite it, then the "Cancel" button should be selected.

If a test has not been configured, then "Generate" cannot be performed for that test. When "Generate" is selected, two files are generated (for most tests), based on the information that was configured. For a more detailed description of the "Generate" function, go to Section 3.5. The addition of a check mark to the "Gen" check box in this window indicates that generation is completed. For some tests, such as the Primality Test and SHA-1 Test, selecting "Generate" will result in a somewhat lengthy generation process, depending on how the test was configured. *Only after generation is complete will a check mark appear in the "Gen" box.*

When both "Conf" and "Gen" check boxes are marked, and a response file has been received, should the "Verification" button be selected. For some tests, this will only require the comparison of values in the facts and response files, and will be relatively quick. However, for the Generation of PQG Values Test, this process will be rather lengthy. *Once again, a check mark will appear in the "Ver" box when verification has been completed.* More information on the "Verification" function may be found in Section 3.6. Note: "Verification" can be run anytime there is a response file and a facts file (where one is necessary) in the same directory. This is important to mention, because if for some reason the original configuration information is overwritten, this would not necessitate the generation of new facts and request files, or require the DUT to create a new response file.

3.2.5 Configuration Window

When the "Configure" button is selected from a Test window, the Configuration window appears. It allows the tester to select different options, based on information pertaining to a particular validation. The configuration information is then used in the generation of request and facts files.

3.2.6 Long Job Status Window

This window will appear during the generation process of the Primality Test, as well as during the generation of PQG values, when this is selected from the Utilities menu item. It allows a tester to cancel this lengthy process, which will take effect once the next prime/non-prime or pqg set has been generated. A progress indicator in this window shows the current counter value for the value being generated (e.g., counter value for generating a pqg set as defined in FIPS 186, Appendix 2.2), the current index of how many values have been generated, as well as the total number of values to be generated.

3.3 New Validation

Before creating a new validation, the DSSVS user should have the following information from the vendor:

• the information listed in Section 1.4, items 1 through 3,

• a list of which modulus sizes are to be tested,

• a list of which tests are to be executed,

• whether or not the DUT can generate X values,

• whether the DUT or DSSVS is the source of PQG triples for key and signature generation tests,

• the pseudorandom number generation method used by the DUT to generate X and K values, and

• for the SHA-1 implementation, whether the DUT can correctly hash messages of any bit-length ($< 2^{64}$ bits), or just byte-oriented messages.

The steps to be taken by the tester to create a new validation are as follows:

1. From the “File” menu, select “New”; enter vendor and DUT information and select a filename for the validation information file; click “OK” in next popup window to continue or “Cancel” to abort.

2. If OK is chosen, then the name of the “.inf” file appears in the titlebar of the main window, and the DSS Validation Suite window appears in the main window.

3. At any time once a validation information file is opened, it can be viewed and modified by selecting “Validation Info” under the “File” menu.

4. If the tester wishes to maintain request, facts, response, and results files in a directory other than the one contain the dssvs executable, then at this time “Open” should be chosen from the “File” menu, and the desired “.inf” file should be opened (even if it’s the same one that was just created). This allows Windows to recognize this other directory as the working directory.

3.4 Configure a Test

Before configuring any test parameters, the user should first create a new validation or open one that already exists (Section 3.3). From the DSS Validation Suite window, the user selects a test to configure. Clicking on the appropriate button, the Test window appears. From here, the “Configure” button should be selected. At that time, a Configuration window will appear. The appropriate selections should be made, in order to configure the test for a particular DUT.

Once configuration is completed, and the “OK” button is selected, then the Test window is activated, and the “Conf” box is checked. The next step is to generate request and facts files (Section 3.5).

Note: Once the "OK" button has been selected in the configuration window, the configuration information is recorded and the corresponding check box is noted in the test window. ***If any configuration information had been saved previously, then it will be overwritten! If the tester wishes to just view the old configuration information, and not overwrite it, then the "Cancel" button should be selected.***

- Primality Test

Selection:	-Modulus sizes to be tested

- PQG Generation Test

Selection:	-Modulus sizes to be tested

- Key Generation Test

Selections:	-Pseudorandom number generation method to be used by the DUT to generate x values (SHA-1, DES, ANSI X9.17) -Source of PQG triples used in generating the key pairs (DUT, DSSVS). -Modulus sizes to be tested

- Signature Generation Test

Selections:	-Pseudorandom number generation method to be used by the DUT to generate x and k values (SHA-1, DES, ANSI X9.17); irrelevant if DSSVS provides x values, or if DUT does not provide XSEED, XKEY, and KKEY values. -Source of PQG triples used in generating the key pairs (DUT, DSSVS). -Source of x values (DUT, DSSVS). -Modulus sizes to be tested.

- Signature Verification Test

Selections:	-Modulus sizes to be tested -Pseudorandom number generation method to be used by the DUT to generate x values (SHA-1, DES, ANSI X9.17)

- Verification of Received PQG Values Test

Selections:	-Modulus sizes to be tested

- SHA-1 Test

Selection: -DUT designed to correctly hash different types of messages (bit-oriented, byte-oriented)

3.5 Generate Request File (and Facts File)

When a user is ready to generate a request file for a test, one should first check that there is a "pqg_file" located in the working directory, and that it in fact contains the correct values.

- Primality Test

  Time: **Very** lengthy
  Note: Depending on the PC used to run the DSSVS, it will take several hours to generate this file. The number of values generated is calculated by the DSSVS, based on the quantity and value of moduli selected. The facts file records each number generated, along with a "P" or "F" to indicate whether the value should pass or fail a primality test.

  Below is a partial list that shows how many values per modulus size that the DSSVS will generate, based on the number of values of modulus sizes chosen. For any combination of modulus sizes chosen, the DSSVS will determine the proper number for each modulus size. This determination is made based on the amount of time it takes to generate a pqg set for each modulus size (see the note in section 3.7). This method is also used in the PQG Generation Test and Verification of Received PQG Values Test to determine how many values to generate/test.

Modulus Sizes	***Number of values to be generated/tested per modulus size***			
	All sizes selected	**512, 768, 1024**	**960, 1024**	**1024 only**
512	15	15	--	--
576	15	--	--	--
640	13	--	--	--
704	9	--	--	--
768	7	15	--	--
832	6	--	--	--
896	4	--	--	--
960	3	--	7	--
1024	3	7	6	10

- PQG Generation Test

  Time: Very quick
  Note: The request and facts files indicate the number of PQG sets to be generated for each identified modulus size. See the table under "Generate Request File: Primality Test" above for an idea of how many values will have to be generated by the DUT.

- Key Generation Test

  Time: Very quick
  Note: The request file indicates that the DUT must generate 10 key pairs for each modulus size (this number is fixed). If the test was configured for the DSSVS to provide PQG sets to the DUT, then one PQG set is pulled from "pqg_file" for each modulus size. The request file also indicates the pseudorandom number generation method that the tool expects the DUT to use in generating a private key, x. (No facts file is generated for this test.)

- Signature Generation Test

  Time: Very quick
  Note: One key pair is generated per modulus size, and 10 pseudorandom messages are generated per modulus size; all of these are placed in the request file. (No facts file is generated for this test.)

- Signature Verification Test

  Time: Quick
  Note: For each modulus size selected, 3 XY key pairs are generated by the DSSVS. For each key pair, a number of messages and signatures will be generated; this number depends on the number of modulus sizes selected. Regardless of the number of modulus sizes chosen, the DUT will be required to perform on the order of 100 verifications, according to the chart below:

# Mod sizes selected	***# Key pairs/mod size***	***# Signatures/key pair***	***Total # Signatures***
1	3	36	108
2	3	18	108
3	3	12	108
4	3	9	108
5	3	7	105
6	3	6	108
7	3	5	105
8	3	4	96
9	3	4	108

  Some of the messages in the request file are paired with "bad" signatures. The facts file indicates what the proper result should be for each signature verification, "P" or "F".

- Verification of Received PQG Values Test

  Time: Quick-to-several minutes
  Note: PQG values are selected from "pqg_file", and some of the sets are modified so that an attempt to verify the values will fail. The number of values per modulus size is determined by the number and value of modulus sizes selected. See the table under "Generate Request File: Primality Test" above for an idea of how many values will have to be generated by the DUT. Generating the facts and request files may take a few minutes due to the tool generating large "bad" primes (i.e., values for p that are not prime). *In some instances, there may be 15 pqg sets tested for a particular modulus size - therefore, the pqg_file should have at least 15 pqg sets per modulus size.*

● SHA-1 Test

Time: Lengthy
Note: Three different sets of values for SHS testing are generated. The request file contains various messages that are to be hashed. The facts file contains the correct message digests that were generated on the messages in the request file.

3.6 Verify Response File

Before verifying a response file for a particular test, the tester should first check that the response file and associated facts file (if one was generated by the DSSVS) are both in the working directory. In the DSS Validation Suite window, a test should be selected. The Test window then indicates whether or not the results have been verified. A response file may be verified multiple times, and whenever verification is complete, the "Ver" checkbox is marked. Information on verification results is stored within the validation information file, and this is used to update the results file (.out) when the "Generate Results File" item is chosen under the "Utilities" menu.

This section describes how the DSSVS performs verifications of results for each test. The amount of time to perform verification depends on how many modulus sizes were selected, and whether the DSSVS has to perform re-calculations or just simply compare stored values. Note that if there are a large number of failures to verify correctly, or there are *no* successful verifications, then this may indicate that the wrong facts file or response file is being used, or the response file is formatted incorrectly. Please see Appendix D and the sample files to make such a determination.

● Primality Test

Time: Quick
Verification method: DSSVS compares results ("P" or "F") in the "prime.fax" and "prime.rsp" files.
Note: In the "prime.rsp" file, a "P" indicates that a number was determined by the DUT to be prime, and an "F" indicates that the DUT found the number to be non-prime. The DSSVS will accept a response as correct if it is either the same as the stored value (e.g., P/P, F/F), or if the response is non-prime ("F"), when the stored value is prime ("P"). This later result is acceptable because a DUT may be using a more robust primality test than the one found in Appendix 2.1 of the DSS.

(When a results file is generated, under the Primality Test there will be a line stating "suspect results on these entries: # #..." The numbers (#) reflect the particular results which indicated "F" when the DSSVS indicated "P". The probably that this will occur is extremely small. However, if this does occur, then the tester should determine what primality test is being used by the DUT, and contact NIST.)

Each time the response is "P" when the stored value is "F", then during verification a message box will indicate "Error: No match, prime number *n*", where it is the *n*th tested value in the entire file (not in the particular modulus size). In order to continue, the "OK" button must be selected. The final results will indicate how many prime numbers were correctly identified.

- PQG Generation Test

Time: Lengthy
Verification method: DSSVS re-calculates values submitted in "pqg.rsp"
Note: This process can take a while, because the DSSVS goes through the pqg generation process defined in Appendix 2.2 of the DSS. If one of the pqg sets is determined to be incorrect, then a message box will be displayed, indicating "Error: Bad pqg set #*n*", where it is the *n*th pqg set in the entire file (not in the particular modulus size). In order to continue, the "OK" button must be selected. The final results will indicate how many sets passed the test.

*Once again, this test shall not be performed if the DUT cannot provide the DSSVS with values for SEED, counter, and h.

- Key Generation Test

Time: Several Minutes
Verification Method: DSSVS re-calculates x values (where possible) and y values submitted in "xy.rsp"
Note: The DSSVS automatically determines how verification will be performed, based on information in "xy.rsp":

1) <u>xkey and xseed both present</u>: DSSVS re-calculates the value for x. In addition, y is re-calculated as $y=g^x modp$. If a bad x value is found, then a message box appears, indicating "Error: Not Verified (Bad X): xy pair number <*n*>".

2) <u>xkey and xseed are NOT present</u>: DSSVS only re-calculates $y=g^x modp$; x cannot be re-calculated.

In either case, if a bad y value is found, then a message box appears, indicating "Error: Not Verified (Bad Y): xy pair number <*n*>". In each case, *n* is the *n*th xy pair in the entire file (not in the particular modulus size). The final results will indicate how many key pairs passed the test.

When XKEY and XSEED are included in the response file, and this test is passed, the validation certificate issued by NIST will reflect that either 1) both x and y were tested, or 2) just y was tested.

- Signature Generation Test

  Time: Quick-to-several minutes
  Verification Method: DSSVS re-calculates signatures (where possible), or else uses y to verify a signature.
  Note: There are three possible scenarios for verification, based on what data is provided in the response file:

  1) If both an x and KKEY value are present, then a signature can be re-calculated by the DSSVS.
  2) If x (but not KKEY) is present, then the public key, y, is re-calculated using x, and y is then used to verify the signature.
  3) If neither x nor KKEY is available, then the y value in the "xy.rsp" file is used to verify the signature.

  When a signature is incorrect, it is indicated by a message box: "Error: Bad Signature <*n*>", where *n* is the *n*th signature in the entire file (not in the particular modulus size). The final results will indicate how many signatures were verified to be generated correctly.

- Signature Verification Test

  Time: Quick
  Verification Method: DSSVS compares results ("P" or "F") in the "versig.fax" and "versig.rsp" files.
  Note: If a result in the response file does not match the stored value in the facts file, then a message box indicating "Error: Not verified: Sig number *n*" is displayed, where *n* is the *n*th signature in the entire file (not in the particular modulus size). The final results will indicate how many signature verification passes and failures were correctly determined.

- Verification of Received PQG Values Test

  Time: Quick
  Verification Method: DSSVS compares results ("P" or "F") in the "verpqg.fax" and "verpqg.rsp" files.
  Note: If a result in the response file does not match the stored value in the facts file, then a message box indicating "Error: Not verified, pqg set number *n*" is displayed, where *n* is the *n*th pqg set in the entire file (not in the particular modulus size). The final results will indicate how many pqg set verification passes and failures were correctly determined.

- SHA-1 Test

  Time: Quick
  Verification Method: DSSVS compares message digests in the "sha.fax" and "sha.rsp" files.

Note: For each of the three SHS tests, a stored message digest value is compared with its corresponding value in the response file. If a message digest does not compare, then a message box indicating "Error: Failed SHS Type *n* Test" is displayed, where *n* is one of the three SHS tests. The final results will indicate whether each of the three tests was passed or failed.

3.7 Supplemental Files

The DSSVS tool includes several supplemental files which may be used when conducting tests. First, there is a file containing a series of pqg values. There are multiple sets of these values for each modulus size, from 512 to 1024 bits. Each set includes the following values: p, q, g, h, SEED, and counter. The latter three values are needed to regenerate the first three values. This file may be used by the DSSVS when generating request files for the Key Generation, Signature Generation, Signature Verification, and Verification of Correct p, q, and g tests. At any time, the "Generate PQG" item under the "Utilities" menu can be selected to generate a new file of pqg values. However, the DSSVS user should be aware that this can be a very time consuming task, depending on the number of modulus sizes selected, and the number of pqg sets to be generated for each size. It is possible to cancel this process after it has begun, however this will take effect only upon completion of the pqg set being generated.

Note: Below is an example of the average time to generate one pqg set, using Version 2.2 of the DSSVS. The values were generated using a PentiumPro, 200MHz PC. The time is the average time to generate one pqg set, based on the generation of 10 pqg sets per modulus size.

Modulus Size (bits)	*Avg.Time/PQG Set (minutes)*
512	*0.8*
576	*1.3*
640	*2.0*
704	*3.3*
768	*3.4*
832	*4.7*
896	*5.7*
960	*8.4*
1024	*11.1*

In addition to the file of pqg values, a set of example test files (information, request, facts, response, and output files) is included with the DSSVS, so that a user can test the full functionality of the tool.

4. MESSAGE FORMATS

Information that is conveyed between the DSSVS and the DUT will be in the form of messages. Request messages are generated by the DSSVS; response messages are generated by the DUT vendor. Request and response messages shall have both header and data portions. The header portion shall indicate one of the four options and whether the information in the data portion is a request or response (e.g., SHS Request Type 1, DSS/APP.S Response Type 3). The delimiters H> and <H shall be used to indicate the beginning and ending of the header portion. The data portion shall consist of ASCII characters and shall be terminated with an ASCII ETX character (^). In addition, ^ will be used to separate keys and data within messages. Other white space characters, such as blank or line feed, may also occur in messages. In the data portion of a message, one or more white space characters must separate two data strings that are not separated by ^. The delimiters D> and <D shall be used to indicate the beginning and ending of the data portion.

4.1 Conventions

The following conventions are used in the data portion of messages between the DSSVS and the DUT:

1. Integers: integers will be unsigned and will be represented in decimal notation or hexadecimal notation.

2. Decimal characters: the ASCII decimal characters to be used are the characters 0-9.

3. Hexadecimal characters: the ASCII hexadecimal characters to be used are the ASCII characters 0-9 and A-F (or a-f), which represent 4-bit binary values.

4. Space characters: In the following descriptions, the notation ◊ represents one or more ASCII white space characters, such as spaces, line feeds, and tabs. However, ◊ is not a legal character in messages. In parsing data in a message, the recipient should read through all characters except 0-9, A-F, a-f, and ^.

4.2 Message Data Types

The following data types are used in messages between the DSSVS and the DUT:

1. Digest: a digest is a string of 40 hexadecimal characters that represents a 160-bit string. The characters must be contiguous.

2. Decimal integers: a decimal integer has the form

    dddd ... dd

where each 'd' represents a decimal character (0-9); one or more characters are present. The characters must be contiguous.

3. Hexadecimal integers: a hexadecimal integer has the form

hhhh ... hh

where each 'h' represents a hexadecimal character (0-9, A-F, a-f); at least two characters are present. The characters must be contiguous.

4. Compact strings: bit strings will be represented in compact form. A compact string has the form

$z \diamond b \diamond n_1 \diamond n_2 \diamond \ldots \diamond n_z$

where z is a decimal integer greater than zero that represents the number of n_i, b is either 0 or 1, and each n_i is a decimal integer representing a positive number. The length of the compact string is given by the summation of the n_i.

** *Note: Code for manipulating compact strings may be found in the "demoutil.c" file, in the "source" directory. The function "hashstring()" can be used to convert a compact string into a bit representation that is used for hashing. The function "genstr()" generates a compact string on the fly, rather than converting from a given hex value. If the vendor requests a conversion utility for compact strings, the vendor may be provided with "demoutil.c".*

The compact string is interpreted as the representation of the bit string consisting of b repeated n_1 times, followed by 1-b repeated n_2 times, followed by b repeated n_3 times, and so on. Example: suppose

$M = 5 \diamond 1 \diamond 7 \diamond 13 \diamond 5 \diamond 1 \diamond 2$

where z = 5 and b = 1. Then the compact string M represents the bit string

1111111000000000000011111011

where 1 is repeated 7 times, 0 is repeated 13 times, 1 is repeated 5 times, 0 is repeated 1 time, and 1 is repeated 2 times.

4.3 Syntax for the SHS Option

For the SHS Option, there are three request types and three corresponding response types.

4.3.1 SHS Type 1

1. SHS Request Type 1, which supplies 1024 messages of varying lengths, shall have the format

$M_1 \wedge M_2 \wedge ... \wedge M_{1024} \wedge$

where each M_i is a compact string of length i for $1 \leq i \leq 1024$.

In cases where the DUT is only to be tested for byte-oriented messages, then 128 messages of varying lengths are supplied, and thus SHS Request Type 1 has the format

$M_1 \wedge M_2 \wedge ... \wedge M_{128} \wedge$

where each M_i is a compact string of length 8*i for $1 \leq i \leq 128$.

2. SHS Response Type 1, which contains 1024 message digests for the 1024 messages from SHS Request Type 1, shall have the format

$D_1 \wedge D_2 \wedge ... \wedge D_{1024} \wedge$

In cases where the DUT is only to be tested for byte-oriented messages, then 128 message digests corresponding with the messages from SHS Request Type 1 are supplied by the DUT, and shall have the format

$D_1 \wedge D_2 \wedge ... \wedge D_{128} \wedge$

Whether there are 128 or 1024 message digests, each D_i is a 160-bit digest of the message M_i, generated using SHA-1.

4.3.2 SHS Type 2

1. SHS Request Type 2, which supplies 100 messages of length > 1024, shall have the format

$M_0 \wedge M_1 \wedge ... \wedge M_{99} \wedge$

where each M_i is a compact string of length between 1032 and 102408 bits for a DUT hashing byte-oriented data, and between 1025 and 102401 bits for a DUT hashing data of an arbitrary bit length.

2. SHS Response Type 2, which contains 100 message digests for the 100 messages from SHS Request Type 2, shall have the format

D_0 ^ D_1 ^ ... ^ D_{99} ^

where each D_i is a 160-bit digest of the message M_i.

4.3.3 SHS Type 3

1. SHS Request Type 3, which supplies a seed message, shall have the format

   M ^

   where M is a compact string of length 420.

2. SHS Response Type 3, which contains 100 message digests using the seed from SHS Request Type 3, shall have the format

   D_0 ^ D_1 ^ ... ^ D_{99} ^

   where each D_i is a 160-bit message digest.

4.4 Syntax for the DSS Tests

Using the correct syntax for creating the DSS test response files is important for the correct verification of those results by the DSSVS. Except for the modulus size (all .rsp files), and counter value (in "pqg.rsp" and "verpqg.rsp"), which are in decimal format, all number values must be presented in hexadecimal format. Three of the response files ("prime.rsp", "versig.rsp", and "verpqg.rsp") include a Pass/Fail result in the form of "P" or "F". Each modulus size is indicated by "[mod=###]". All alphanumeric values are preceded by "*tag*= ", where there is a blank space between the equals sign and the subsequent value (There is no blank space in the case of the modulus size indicator.). Please see Appendices C and D, as well as the electronic sample response files, for the correct syntax of DSS test values.

4.5 File Formats

The format of request files can be found in Appendix C, and the format of response files can be found in Appendix D. Note that these appendices show what all message types look like in a request or response files. Request files are generated by the DSSVS, while response files must be generated by the vendor/tester, so that they can be properly handled by the DSSVS. Facts files are generated automatically by the DSSVS, and therefore sample facts files are not presented in any appendix to this user's guide.

APPENDIX A: Validation Information File Format

The following is an example validation information file, which can be modified manually, if necessary:

```
[Vendor]
file_name=sample.inf
product_name=DSS Software
version=1.0
impl_type_sw=Yes
impl_type_fw=No
impl_type_hw=No
processor=Intel Pentium; Windows95
vendor_name=Jim's LLC
vendor_street=813 Chestnut Tree Drive
vendor_citystatezip=Annapolis, MD  99999
vendor_contact=Jimbo Jones
vendor_phone=(410) 123-4567
internet_address=jimbo.jones@jimbollc.com

[Primality]
configured=Yes
generated=Yes
verified=Yes
mods=512 768
suspect=19
total=30
matched=30

[PQG]
configured=Yes
generated=Yes
verified=Yes
mods=512 768
total=30
matched=30

[XY]
configured=Yes
generated=Yes
verified=Yes
mods=512 768
x_verified=YES
random_method=SHA-1
pqg_source=DUT
total=20
matched=20

[GenSig]
configured=Yes
generated=Yes
verified=Yes
mods=512 768
random_method=SHA-1
pqg_source=DSSVS
x_source=DSSVS
total=20
```

```
matched=20


[VerSig]
configured=Yes
generated=Yes
verified=Yes
mods=512 768
random_method=SHA-1
total=108
matched=108

[VerPQG]
configured=Yes
generated=Yes
verified=Yes
mods=512 768
total=30
matched=30

[SHA-1]
configured=Yes
generated=Yes
verified=Yes
DataSize=BIT
type1=Passed
type2=Passed
type3=Passed
```

APPENDIX B: Results File Format

The following is an example results file:

```
DSS Validation Suite Results for "DSS Software"

Primality Test:
   Mod sizes selected are: 512 768
   Primality Results: 30 verified out of 30
   Suspect results on these entries: 19

Generation of PQG Test:
   Mod sizes selected: 512 768
   Generation of PQG Results: 30 verified out of 30

Key Generation Test:
   Number of XY sets per mod size: 10
   X and Y tested for correctness
   Random number generator type is: SHA-1
   Mod sizes selected: 512 768
   Generation of XY Results: 20 verified out of 20

Signature Generation Test:
   Random number generator type is: SHA-1
   Mod sizes selected: 512 768
   Signature Generation: 20 verified out of 20

Signature Verification Test:
   Mod sizes selected: 512 768
   Signature Verification: 108 verified out of 108

Verification of Correct Generation of P, Q, and G Test:
   Mod sizes selected: 512 768
   Verification of Correct Generation of PQG: 30 verified out of 30

SHA-1 Test:
   SHA-1 tests are configured for "BIT" oriented implementations
   SHA-1 Test results for:
      Messages of Varying Length: Passed
      Selected Long Messages: Passed
      Pseudorandomly Generated Messages: Passed
```

APPENDIX C: Request File Formats

This appendix shows the formats for the seven request files that can be generated by the DSSVS. These files are created when the “Generate” function is chosen for each corresponding test. As in the actual files, comments below are indicated with a “#” located in the first position of the commented line. In the request files for the DSS tests, information at the beginning of each file lists what options were selected for the test, so that the tester generating the response files will know what information to include in those files.

Note that for the DSS test files, there should be one blank space between each equals sign and the number which follows (except for the modulus size indicators).

C.1 SHS Request File Format (“sha.req”)

All three SHS request types make use of compact strings to express messages, to minimize the file size. The format for compact strings is explained in section 4.2 of this document. *Code for manipulating compact strings may be found in the “demoutil.c” file, in the “source” directory.* Spaces between the compact string and ‘^’ are not included in the actual file.

```
H>SHS Request Type 1<H
D>
compact_string[0]^
     . . .
compact_string[1024]^
<D

H>SHS Request Type 2<H
D>
compact_string[0] ^
     . . .
compact_string[99] ^
<D

H>SHS Request Type 3<H
D>
compact_string ^
<D
```

C.2 Primality Test Request File Format (“prime.req”)

```
#  Configuration information for "DSS Software"
#  Mod sizes selected are: 512 768
[mod=512]
Prime=99dd9a2859b94200927cc5ca34754ad421ce8ea869aefbad3990cf2c0485ecc49bb5a5d9957356
59b0f02328fdd3dc82614c774cfa8b120c4c2ec009cdfc564d
Prime=bffc22229ba35bd4f4650f7b120b96bb057890979d4a7da2b6e71e3dbc634cae1ee33d4e6546f0
669f552b151904e89b553510e994862d6fc88ccb8792df301f
Prime=a3c03f6fc2b64c61063914cbc102eeb9d56c394a6f71022d2c4e1d07f7043eefeeca6ec53de2fe
2a003f551494ba3a7086b952535bb0332723450b75c63296d3
```

```
. . .
[mod=768]
Prime=dfc03ba47720d7fcb374a2a5b75779d372dccf93bbfa855bbd524335b94b2c83b5fbcb25007c2b
77a24ecd5e67ee16c5af585f1216c4bed47030fd2b618f175e84868dc3ca78a5ddd3860302fe4c35a2cb
292e1e9ee02b001217ce60f9802afb
Prime=b1a804675d427b2b29a6d6d61da575e06328e574ba8ccc1d995476f90da2e183a166784edeecc2
ad829a70831d4cfceba08ddabb0c14e3474304643aac0f6b35c05cc4d1aa8eaf8b0c69929e601acda835
f4d5d9082666b237e62d1ac5f838b7
Prime=f94c4f626a389d869bf3d2cbcf9b354acd43d028967e746f75306d89afc61d0736a21b95c78e50
1fd22e54d6c11e1c094fd1590480ac688ed16469d3e59c4c9dba97963e22aec43f2d2228643541469e4c
19ab07a9957d9ec6851e002b1b8de9
. . .
```

C.3 PQG Generation Request File Format (“pqg.req”)

```
#  Configuration information for "DSS Software"
#  Number of PQG sets per mod size:
#  Mod sizes selected: 512 768
[mod=512]
N= 15
[mod=768]
N= 15
```

C.4 Key Generation Request File Format (“xy.req”)

Note that in actuality, the DSSVS tool requires a fixed value of 10 key pairs per modulus size.

```
#  Configuration information for "DSS Software"
#  Number of XY sets per mod size: 10
#  Random number generator type is: SHA
#  Mod sizes selected: 512 768
```

C.5 Signature Generation Request File Format (“gensig.req”)

```
#  Configuration information for "DSS Software"
#  Number of Signatures and Messages per mod size:
#  Random number generator type is: SHA
#  Mod sizes selected: 512 768
[mod=512]
P=8df2a494492276aa3d25759bb06869cbeac0d83afb8d0cf7cbb8324f0d7882e5d0762fc5b7210eafc2
e9adac32ab7aac49693dfbf83724c2ec0736ee31c80291
Q= c773218c737ec8ee993b4f2ded30f48edace915f
G=626d027839ea0a13413163a55b4cb500299d5522956cefcb3bff10f399ce2c2e71cb9de5fa24babf58
e5b79521925c9cc42e9f6f464b088cc572af53e6d78802
XKey= 1234567890123456789012345678901234567890
XSeed= 0000000000000000000000000000000000000000
X= 436f11fbb83ab498016c4942152a83c0090934a2
Msg=c713d10ff3357345ddfbbfe623abdb5f010934677683becd9a9f70863fa3faf6c783c97e450f3e48
d85d2a4f44758db1d13c903969a3c028ed108d44d70c0224162fc6ba2649542170d277a9e1940920cd97
ec63064d492638fe0800887790ba6defc1d9ce392e695d319f0c3360c744acf242fd85dad25e3543dbd1
883a1c51
```

```
Msg=829bb0f3733944b857519a9270313fb42725881f1fa2d3699ab6fcd012af1e80100b8b224fa00ea5
160c6050d35fe809f408b7df0bfdc4df222f64155b2d0de0cc174c7b99c604c95339e86191433bdacd73
c75881441c5782860bb79d0c62097097e9188a039dbbc6b02cdbe433b0c06a3eb0a0bacd556a7493e9cf
0fa49592
. . .
```

C.6 Signature Verification Request File Format (“versig.req”)

```
#  Configuration information for "DSS Software"
#  Random number generator type is: SHA
#  Mod sizes selected: 512 768
[mod=512]
P=bd0669882bed2f2937b74c7a34db1e607b4250915590faebc8939b60ef8e1d53c165b67019c4772e00
beace035db809b059da89d5e9d6557d2332ccd29c2d00b
Q= 9c1bae515c1a397e753ac4f8c3c389721a8ea0bf
G=2d7344dfb83c1db8a23c4139b0dbfcc62df17eb56b4eaab32f2674e650ca079823863e1954a8310808
b25ea8891bf7c6b63262a029a52647557b68d3ce6094d7
Y=7f54905620066405ea62787f4512ddc2ca033106be2040d999af661375506f57f0e3594c660c2cdd21
6e6e167c953112b4bd4893e0672d7b3cf8fe17e5008790
Msg=5ffae3a6029225f5f7deb58060b83516c11a595dedde6eaf33b42b34c464bc4068460def61ff54fe
9f7de1ed27352a53bd9c1281d9e47a5bc9f80a7223ba41666e7cbb7c10bdc27b7a2519cc93c304d16385
9b8501f85156bf42c03f935db72ac9958685e7458923e1cef3557ed8dc480acd8ea23d920c586b88e4d3
eec33743
Sig=5586be9c5254d464ce9822dcdd28752236768e2d5ab5b3e9779eba87b5cf1091f51438750e3fe9e1
Msg=43d10f06aebe6fbec02aed0771bf6bca0e092a811a65c3c36427650f690b50d8ecde92d2d46d027c
46ae7aee032d58e504ba370c5351eb8d2c4abbd855c2e2b3bb4848824ecc7dd91ec5f73fb6a28c45b974
33c8bbd9f782622c3cd8beea37dfe568a7afd3b3d9ed80c8db9443f5ff03678f944c0bd5e0bc3f2462a6
5c5d4675
Sig=26e6b7441de0208f3af15217319bf0aafd913a2688cf59fe0e7423d97dc2d816d5124e800b210eac
```

C.7 Verification of Received PQG Values Request File Format (“verpqg.req”)

```
. . .

[mod=768]
P=abbb6aa9760c7f3671c013c1c712a2ac907f5924ebf7e3a3892c9111263864287850c585253d807654
923184d7e47745ebbf001f1eac5fce88d13bc564d126a56cbff89bddc28d69f1a00013c6bf5a63141cab
adaff1850cebc0c50d40b19581
Q= dbd8daaacc4f27fc05508757259e27e975bfdc2f
G=9ed9ab75abe9450372b95843df66a006a50097c4b6563f377882a3d8bede1b4b7fe055cf4ee62c6bc3
f977ed0553a9953c21bd4901be016a4b7558311e2c409e7ad6cf666a06f217c531a1757adb4669b2e0da
82a0903d71deed11632ef131ca
H=000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000002
Seed= d5014e4b60ef2ba8b6211b4062ba3224e0427eea
c= 162
P=d035088552dbae1ed41fe0374471511b6b190ce1b3b1b79b5d1faf0f11ee876e02a9f66442b4694649
d75cf8185908cf2ac5bda8e49aeab90f0c530ba5643b85fbca7cf3dfbbb85e572421d4123150daa05f97
4d38b8a0776eb0da7176601861
Q= e2867218b7a25fd993d0178a1b0a1ab7bfae3cd1
```

```
G=9248ea5f6f59bf5c114925ad7c53ce2985c9f4c25255dd8be092932f99f950499158ed448f99e61de2
1ed4935e85604c412c8d88bf1f663ed5bb7f3a04ff498ccb9121007850bf46fa38cf35105e1d7fdd25c6e70
H=0000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000002
Seed= c5cc4392f040af1521d166afbad4b75bf634b0e2
c= 153
...
```

APPENDIX D: Response File Formats

This appendix shows the formats for the seven response files that are to be generated by the DUT. The DSSVS expects these files to be in the formats indicated below. When a response file is present for a particular test, and the "Verify" function is chosen for that test, then the DSSVS is able to properly check the correctness of the DUT's test results. As in the actual files, comments below are indicated with a "#" located in the first position of the commented line. Here are some basic guidelines for generating response files:

1) For all response files (except for the SHA-1 test, and for modulus size indicators), there should *always* be a blank space between the equals sign and the value that follows,
2) There should be no blank lines,
3) The first character of all variable names should be in the first column, and should be capitalized (except for the lowercase "c" for the pqg counter value, and the "result" variable),
4) The variable names should be spelled (and have the same case) as specified in the following examples as well as in the sample files accompanying the DSSVS, and
5) There should be no padding after the last character of a value, and each line should be terminated immediately thereafter using a NEWLINE character. (Note that the tool will ***not*** function correctly if CARRIAGE RETURN/LINE FEED is used to terminate a line.)

D.1 SHS Response File Format ("sha.rsp")

Code for manipulating compact strings may be found in the "demoutil.c" file, in the "source" directory.

```
H>SHS Response Type 1<H
D>
digest[1] ^
     . . .
digest[1024] ^
<D

H>SHS Response Type 2<H
D>
digest[0] ^
     . . .
digest[99] ^
<D

H>SHS Response Type 3<H
```

```
D>
digest[0] ^
     . . .
digest[99] ^
<D
```

D.2 Primality Test Response File Format (“prime.rsp”)

```
#  Configuration information for "DSS Software"
#  Mod sizes selected are: 512 768
[mod=512]
Prime=99dd9a2859b94200927cc5ca34754ad421ce8ea869aefbad3990cf2c0485ecc49bb5a5d9957356
59b0f02328fdd3dc82614c774cfa8b120c4c2ec009cdfc564d
result= P
Prime=bffc22229ba35bd4f4650f7b120b96bb057890979d4a7da2b6e71e3dbc634cae1ee33d4e6546f0
669f552b151904e89b553510e994862d6fc88ccb8792df301f
result= P
Prime=a3c03f6fc2b64c61063914cbc102eeb9d56c394a6f71022d2c4e1d07f7043eefeeca6ec53de2fe
2a003f551494ba3a7086b952535bb0332723450b75c63296d3
result= P
Prime=be687acfdf48786effc6e9542ba9ef88b242afda1ee5d4bede147ded63118010e245e6c6280591
1bcf3bdf67295cfd9719fa8f53665e7f2a8793a64ac919fda1
result= P
Prime=f92b83f33601ba3926fc986b433cc34971fc1626f9c1f2ae7361457560155555ad22d0fb8333755
57ed4c0fa918a5e871e222615c20a5ffec909f7a196e5919f9
result= F
. . .
```

D.3 PQG Generation Response File Format (“pqg.rsp”)

```
[mod=512]
P=824c1c23066aeafa74b40cbfcb2ffd47baed14e5edc3c929f81e2b2fbdfec178939b8aebac70869e4e
ad77653ec497837ea1a201f878fc72ec609eb66eed3a9b
Q= e4b5d099504cf0f208ad237b1c3af79f31cf19bf
G=493509dd3f575e14306aa5a3106c14dd528f2966d9dc8d2136a1b39f9ce87d533bc756cab9faf27a82
8202845d44401ba453db62a28c02b42910c64fa1a9e3f2
Seed= 4b5f92cfde7822b28dfe1c264ec085a3b7df82ed
H=0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000002
c= 123
P=fa78f57cbbfd802d1fcb743eed0cabf2648bee5d89e5b067ad8c64c55ff0b3c2f755b60a5340fbccb2
3c0adc8158724ec8dcd580f8c4c16f8d924c5bc1ae76cb
Q= bf4dc8f35bc250d2f8d95f36c95841b2616f0533
G=7b669fa6609e12fab02e810a494e49db20b9bf24aa7765d5f6ac100a4801d1d667c11a7a77ea7c0cd0
ec83b9ab844bc3d9a3c5d8910019ecdd2b1c8691649f7f
Seed= 4b5f92cfde7822b28dfe1c264ec085a3b7df8311
H=0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000002
c= 109
...
```

D.4 Key Generation Response File Format (“xy.rsp”)

```
#  Configuration information for "DSS Software"
#  Number of XY sets per mod size: 10
#  Random number generation method is: SHA-1
#  Mod sizes selected: 512 768
[mod=512]
P=bd0669882bed2f2937b74c7a34db1e607b4250915590faebc8939b60ef8e1d53c165b67019c4772e00
beace035db809b059da89d5e9d6557d2332ccd29c2d00b
Q= 9c1bae515c1a397e753ac4f8c3c389721a8ea0bf
```

```
G=2d7344dfb83c1db8a23c4139b0dbfcc62df17eb56b4eaab32f2674e650ca079823863e1954a8310808
b25ea8891bf7c6b63262a029a52647557b68d3ce6094d7
XKey= c92190894f57ad80cdf653d0b48319dc3a373acc
XSeed= aedfc3d13bda1701f0cb2f5fbb61e426a6da792a
X= 1be8f6ac9ca1026b3a8cf459350c86140801a34a
Y=1988427903e2d627c9acf47e4f73cc91fe732cda1a387aba8123f59a5e960326e78d3cd38d483d9022
38ee3739e104ebc051e07313e1417b5c9757e35edc9fef
XKey= e50a8735ebf8afec08834829e98f9ff04238de17
XSeed= aedfc3d13bda1701f0cb2f5fbb61e426a6da792a
X= 57bce9a497017dd68cc683f4361fe03f6b56c615
Y=a1c10720cbb29ff075c64cb39eb2022f566cc71e5fb17c4cc59b4a640936130e33d53ba922fe2d389d
3a1adce25733d96ef58d9c9f80c1fa68012bc066e44cbc
...
```

D.5 Signature Generation Response File Format ("gensig.rsp")

Note that if the DUT can export the value for KKEY, then it should be placed in this file, between the G value and the Msg value.

```
#  Configuration information for "DSS Software"
#  Number of Signatures and Messages per mod size: 10
#  Random number generator type is: SHA
#  Mod sizes selected: 512 768
[mod=512]
P=8df2a494492276aa3d25759bb06869cbeac0d83afb8d0cf7cbb8324f0d7882e5d0762fc5b7210eafc2
e9adac32ab7aac49693dfbf83724c2ec0736ee31c80291
Q= c773218c737ec8ee993b4f2ded30f48edace915f
G=626d027839ea0a13413163a55b4cb500299d5522956cefcb3bff10f399ce2c2e71cb9de5fa24babf58
e5b79521925c9cc42e9f6f464b088cc572af53e6d78802
X= 436f11fbb83ab498016c4942152a83c0090934a2
Msg=c713d10ff3357345ddfbbfe623abdb5f010934677683becd9a9f70863fa3faf6c783c97e450f3e48
d85d2a4f44758db1d13c903969a3c028ed108d44d70c0224162fc6ba2649542170d277a9e1940920cd97
ec63064d492638fe0800887790ba6defc1d9ce392e695d319f0c3360c744acf242fd85dad25e3543dbd1
883a1c51
Sig=2e5564fc37352b7eac6119d1d37d3c9e5124801e13f0891adcbf9fbcb85dba49d64a2fdc5821a5dc
Msg=829bb0f3733944b857519a9270313fb42725881f1fa2d3699ab6fcd012af1e80100b8b224fa00ea5
160c6050d35fe809f408b7df0bfdc4df222f64155b2d0de0cc174c7b99c604c95339e86191433bdacd73
c75881441c5782860bb79d0c62097097e9188a039dbbc6b02cdbe433b0c06a3eb0a0bacd556a7493e9cf
0fa49592
Sig=a61ef89219b440779e81950a93641600f6966d6a72d23f357065a79a4b547ccafff2f4ac662544c1
. . .
```

D.6 Signature Verification Response File Format ("versig.rsp")

```
#  Configuration information for "DSS Software"
#  Random number generator type is: SHA
#  Mod sizes selected: 512 768
[mod=512]
P=bd0669882bed2f2937b74c7a34db1e607b4250915590faebc8939b60ef8e1d53c165b67019c4772e00
beace035db809b059da89d5e9d6557d2332ccd29c2d00b
Q= 9c1bae515c1a397e753ac4f8c3c389721a8ea0bf
G=2d7344dfb83c1db8a23c4139b0dbfcc62df17eb56b4eaab32f2674e650ca079823863e1954a8310808
b25ea8891bf7c6b63262a029a52647557b68d3ce6094d7
```

```
Y=7f54905620066405ea62787f4512ddc2ca033106be2040d999af661375506f57f0e3594c660c2cdd21
6e6e167c953112b4bd4893e0672d7b3cf8fe17e5008790
Msg=5ffae3a6029225f5f7deb58060b83516c11a595dedde6eaf33b42b34c464bc4068460def61ff54fe
9f7de1ed27352a53bd9c1281d9e47a5bc9f80a7223ba41666e7cbb7c10bdc27b7a2519cc93c304d16385
9b8501f85156bf42c03f935db72ac9958685e7458923e1cef3557ed8dc480acd8ea23d920c586b88e4d3
eec33743
Sig=5586be9c5254d464ce9822dcdd28752236768e2d5ab5b3e9779eba87b5cf1091f51438750e3fe9e1
result= P
Msg=43d10f06aebe6fbec02aed0771bf6bca0e092a811a65c3c36427650f690b50d8ecde92d2d46d027c
46ae7aee032d58e504ba370c5351eb8d2c4abbd855c2e2b3bb4848824ecc7dd91ec5f73fb6a28c45b974
33c8bbd9f782622c3cd8beea37dfe568a7afd3b3d9ed80c8db9443f5ff03678f944c0bd5e0bc3f2462a6
5c5d4675
Sig=26e6b7441de0208f3af15217319bf0aafd913a2688cf59fe0e7423d97dc2d816d5124e800b210eac
result= P
. . .
```

D.7 Verification of Received PQG Values Response File Format ("verpqg.rsp")

```
. . .

[mod=768]
P=abbb6aa9760c7f3671c013c1c712a2ac907f5924ebf7e3a3892c9111263864287850c585253d807654
923184d7e47745ebbf001f1eac5fce88d13bc564d126a56cbff89bddc28d69f1a00013c6bf5a63141cab
adaff1850cebc0c50d40b19581
Q= dbd8daaacc4f27fc05508757259e27e975bfdc2f
G=9ed9ab75abe9450372b95843df66a006a50097c4b6563f377882a3d8bede1b4b7fe055cf4ee62c6bc3
f977ed0553a9953c21bd4901be016a4b7558311e2c409e7ad6cf666a06f217c531a1757adb4669b2e0da
82a0903d71deed11632ef131ca
H=000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000002
Seed= d5014e4b60ef2ba8b6211b4062ba3224e0427eea
c= 162
result= P
P=d035088552dbae1ed41fe0374471511b6b190ce1b3b1b79b5d1faf0f11ee876e02a9f66442b4694649
d75cf8185908cf2ac5bda8e49aeab90f0c530ba5643b85fbca7cf3dfbbb85e572421d4123150daa05f97
4d38b8a0776eb0da7176601861
Q= e2867218b7a25fd993d0178a1b0a1ab7bfae3cd1
G=9248ea5f6f59bf5c114925ad7c53ce2985c9f4c25255dd8be092932f99f950499158ed448f99e61de2
1ed4935e85604c412c8d88bf1f663ed5bb7f3a04ff498ccb9121007850bf46fa38cf35105e1d7fdd25c6
e70352b911eb85b91f1cbcac3b
H=000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000002
Seed= c5cc4392f040af1521d166afbad4b75bf634b0e2
c= 153
result= P
. . .
```

APPENDIX E: Description of the SHS Type 3 Test

This test determines whether the DUT can compute message digests for messages that are generated using a given seed, which is provided in "sha.req". A sequence of 100 message digests is generated by the DUT using this seed. The DUT portion of the testing procedure is as follows:

The DUT:

1. Obtains SHS Request Type 3 message M (416 bits) from the "sha.req" file (this is the "seed").

2. Performs the following test, using M as input:

```
procedure testSHS(M,D[0], . . . D[99])
    string M,D[0], . . . D[99];
    {
    integer i, j, a;
    for j = 0 to 99 do
        {
        for i = 1 to 50000 do
                {
                for a = 1 to (j/4*8 + 24) do M := M || '0';      /*'0' is the binary zero bit. */
                M := M || i;                      /* Here, the value for 'i' is expressed as a 32-bit word
                                                  and concatenated with 'M'. The first bit concatenated
                                                  with 'M' is the most significant bit of this 32-bit word.
                                                  */
                M := SHA(M);
                }
        D[j] := M;
        }
    }
```

*NOTE: In the above procedure, || denotes concatenation. Also, M || i denotes appending the 32-bit word representing the value 'i', as defined in section 2 of the SHS. Within the procedure, M is a string of variable length, determined by the DSSVS; its initial value is assumed to be input. Together, the initial length of 416 bits and the expression "j/4*8 + 24" (where j/4 is integer division) ensure that messages will be of a byte length. Each element of the resulting sequence {D[j]} should be 160 bits in length.*

3. Forwards the resulting 100 message digests stored in D[0], . . . D[99] as a sequence in SHS Response Type 3 with D_i = D[j]. This is the last section of the "sha.rsp" file.